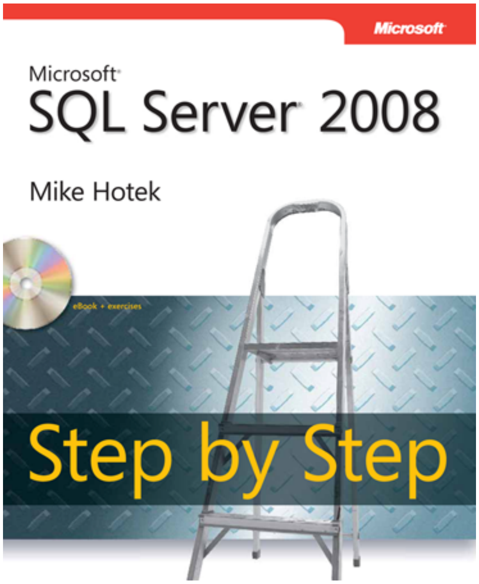

Microsoft® SQL Server® 2008 Step by Step

Mike Hotek

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/MSPress/books/12859.aspx

9780735626041

Table of Contents

Chapter 14

Triggers

After completing this chapter, you will be able to

- Create DML triggers
- Create DDL triggers

Triggers provide a means to allow you to automatically execute code when an action occurs. Two types of triggers are available in Microsoft SQL Server 2008: DML and DDL. In this lesson, you will learn how to create DML triggers that execute when you add, modify, or remove rows in a table. You will also learn how to create DDL triggers that execute when DDL commands are executed or users log in to an instance.

DML Triggers

Although functions and stored procedures are stand-alone objects, you can't directly execute a trigger. *DML triggers* are created against a table or a view, and are defined for a specific event—*INSERT, UPDATE,* or *DELETE*. When you execute the event a trigger is defined for, SQL Server automatically executes the code within the trigger, also known as "firing" the trigger.

The generic syntax for creating a trigger is:

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }
```

When a trigger is defined as AFTER, the trigger fires after the modification has passed all constraints. If a modification fails a constraint check, such as a check, primary key, or foreign key, the trigger is not executed. AFTER triggers are only defined for tables. You can define multiple AFTER triggers for the same action.

A trigger defined with the INSTEAD OF clause causes the trigger code to be executed as a replacement for *INSERT, UPDATE,* or *DELETE*. You can define a single INSTEAD OF trigger for a given action. Although INSTEAD OF triggers can be created against both tables and views, INSTEAD OF triggers are almost always created against views.

Regardless of the number of rows that are affected, a trigger only fires once for an action.

As explained in Chapter 10, “Data Manipulation,” SQL Server makes a pair of tables named inserted and deleted available when changes are executed.

In the following exercise, you will create a DML trigger that populates the FinalShipDate column in the Orders.OrderHeader table when the ShipDate column has been populated for all rows in the Orders.OrderDetail table for an OrderID.

Create a DML Trigger

1. Execute the following code against the SQL2008SBS database (the code is from the Chapter14\code1.sql file in the book’s accompanying samples):

```
CREATE TRIGGER tiud_orderdetail ON Orders.OrderDetail
FOR INSERT, UPDATE, DELETE
AS

UPDATE a
SET a.FinalShipDate = c.FinalShipDate
FROM Orders.OrderHeader a INNER JOIN
    (SELECT od1.OrderID, MAX(od1.ShipDate) FinalShipDate
    FROM Orders.OrderDetail od1 INNER JOIN
        (SELECT od2.OrderID
        FROM Orders.OrderDetail od2 INNER JOIN inserted i ON od2.OrderID = i.OrderID
        WHERE od2.ShipDate IS NOT NULL
        EXCEPT
        SELECT od3.OrderID
        FROM Orders.OrderDetail od3 INNER JOIN inserted i ON od3.OrderID = i.OrderID
        WHERE od3.ShipDate IS NULL) b
    ON od1.OrderID = b.OrderID
    GROUP BY od1.OrderID) c
ON a.OrderID = c.OrderID
GO
```

2. Validate your newly created trigger by setting the ShipDate column for all order detail rows for an order.

In the following exercise, you will create a DML trigger that enforces referential integrity between the SQL2008SBS and SQL2008SBSFS databases.

Create a DML Trigger

1. Execute the following code against the SQL2008SBS database (the code is from the Chapter14\code2.sql file in the book’s accompanying samples):

```
USE SQL2008SBSFS
GO

CREATE TRIGGER tiu_productdocuments ON Products.ProductDocument
FOR INSERT, UPDATE
AS
IF EXISTS (SELECT 1 FROM SQL2008SBS.Products.Product a
            INNER JOIN inserted b ON a.ProductID = b.ProductID)
```

```
BEGIN
    RETURN
END
ELSE
BEGIN
    ROLLBACK TRANSACTION
    RAISERROR('Violation of foreign key',16,1)
END
GO

USE SQL2008SBS
GO

CREATE TRIGGER td_product ON Products.Product
FOR DELETE
AS
IF EXISTS (SELECT 1 FROM SQL2008SBSFS.Products.ProductDocument a
            INNER JOIN deleted b ON a.ProductID = b.ProductID)
BEGIN
    ROLLBACK TRANSACTION
    RAISERROR('You must first delete all documents for this product',16,1)
END
ELSE
BEGIN
    RETURN
END
GO
```

2. Validate your newly created trigger by attempting to insert a document with a ProductID that does not exist.

DDL Triggers

DDL triggers execute under the following circumstances:

- DDL is executed.
- A user logs into an instance.

The general syntax for creating a DDL trigger is as follows:

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier > [ ; ] }

<ddl_trigger_option> ::=
  [ ENCRYPTION ]  [ EXECUTE AS Clause ]

<method_specifier> ::=
  assembly_name.class_name.method_name
```

DDL triggers can be scoped at either the database or instance level. To scope a DDL trigger at the instance level, you utilize the ON ALL SERVER option. To scope a DDL trigger at the database level, you utilize the ON DATABASE option.

The following is an example of a DDL trigger:

```
CREATE TRIGGER tddl_tabledropalterprevent
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
  PRINT 'You are attempting to drop or alter tables in production!'
  ROLLBACK;
```

Note Almost all DDL commands run within the context of a transaction. Since a DDL trigger also runs within the same transaction context, any DDL statement running in the context of a transaction can be rolled back. *ALTER DATABASE* is one of the commands which does not execute in the context of a transaction, because the command affects objects outside of SQL Server that do not obey transactional semantics. Therefore an *ALTER DATBASE* command cannot be rolled back.

The value for the event type is derived from the DDL statement being executed, as listed in Table 14-1.

TABLE 14-1 DDL Trigger Event Types

DDL Command	Event Type
CREATE DATABASE	CREATE_DATABASE
DROP TRIGGER	DROP_TRIGGER
ALTER TABLE	ALTER_TABLE

Event types roll up within a command hierarchy called *event groups*. For example, the CREATE_TABLE, ALTER_TABLE, and DROP_TABLE event types are contained within the DDL_TABLE_EVENTS event group. Event types and event groups allow you to create flexible and compact DDL triggers.

More Info The events and associated event groups that are valid for a DDL triggers can be found in the Books Online article, “Event Groups for Use with DDL Triggers.”

Although DML triggers have access to the inserted and deleted tables, DDL triggers have access to the *EVENTDATA()* function which returns the following XML document that can be queried by using the *value()* method available through XQUERY:

```
<EVENT_INSTANCE>
    <EventType>type</EventType>
    <PostTime>date-time</PostTime>
    <SPID>spid</SPID>
    <ServerName>name</ServerName>
    <LoginName>name</LoginName>
    <UserName>name</UserName>
    <DatabaseName>name</DatabaseName>
    <SchemaName>name</SchemaName>
    <ObjectName>name</ObjectName>
    <ObjectType>type</ObjectType>
    <TSQLCommand>command</TSQLCommand>
</EVENT_INSTANCE>
```

You can retrieve the database, schema, object, and command that you executed, through the following query:

```
SELECT EVENTDATA().value
        ('(/EVENT_INSTANCE/DatabaseName)[1]','nvarchar(max)'),
EVENTDATA().value
        ('(/EVENT_INSTANCE/SchemaName)[1]','nvarchar(max)'),
EVENTDATA().value
        ('(/EVENT_INSTANCE/ObjectName)[1]','nvarchar(max)'),
EVENTDATA().value
        ('(/EVENT_INSTANCE/TSQLCommand)[1]','nvarchar(max)')
```

In the following exercise, you create a DDL trigger to prevent accidentally dropping tables in a production environment.

Create a Database Level DDL Trigger

1. Execute the following code against the SQL2008SBS database (the code is from the Chapter14\code3.sql file in the book's accompanying samples):

   ```
   CREATE TRIGGER tddl_preventdrop
   ON DATABASE
   FOR DROP_TABLE
   AS
       PRINT 'Please disable DDL trigger before dropping tables'
       ROLLBACK TRANSACTION
   GO
   ```

2. Validate your trigger by attempting to drop a table in the SQL2008SBS database.

In the following exercise, you create a logon trigger to limit the number of concurrent connections to a user.

Create an Instance Level DDL Trigger

1. Execute the following code (the code is from the Chapter14\code4.sql file in the book's accompanying samples):

   ```
   CREATE TRIGGER tddl_limitconnections
   ON ALL SERVER
   FOR LOGON
   ```

```
AS
BEGIN
IF (SELECT COUNT(*) FROM sys.dm_exec_sessions
    WHERE is_user_process = 1 AND
        login_name = suser_sname()) > 5

    PRINT 'You are only allowed a maximum of 5 concurrent connections'
    ROLLBACK
END
GO
```

2. Validate your trigger by attempting to create more than five concurrent connections.

Note You have to be careful with a logon trigger, especially one that prevents logging on to the instance. In the exercise above, you had the trigger apply to **all** logins. You should always exclude logins that are members of the sysadmin role, because you do not want to cause a sysadmin to not be able to log in to an instance.

Chapter 14 Quick Reference

To	Do This
Execute code when a DML command is executed	Create a DML trigger
Execute code when a DDL command is executed	Create a DDL trigger